forests

MDPI

Supplementary Material

Heuristic Optimization of Thinning Individual Douglas-Fir

Todd West, John Sessions and Bogdan M. Strimbu *

S1. Heuristic Implementations

S1.1. Preface: 1-opt Flip and 2-opt Exchange Moves

All eight heuristics considered use 1-opt moves to perform optimization. A 1-opt move flips a single tree's cut schedule between thinning and final harvest. If the tree is currently scheduled for thinning, the heuristic instead evaluates what would happen if it was cut at final harvest. Conversely, if a tree is scheduled for final harvest then a 1-opt move results in checking whether it would be preferable to thin it. Performing many 1-opt moves allows a heuristic to vary the percentage of trees selected for thinning as well as the intensity of thinning within each diameter class. For a stand represented by N trees there are N possible 1-opt moves and each run of a heuristic is likely to evaluate each move several times. We also made limited use of 2-opt exchanges and exchange-like patterns of 1-opt moves. A 2-opt exchange picks two trees and swaps their schedules. While an exchange does not vary the number of trees selected for thinning, it can make smaller net changes in the basal area intensity of a thin than a 1-opt flip and frequently reallocates tree selections between diameter classes. Exchanges therefore tend to create smaller objective function changes than flips, a property often associated with increased likelihood of exchanges being accepted as moves. A heuristic search employing both 1-opt and 2-opt moves is thus more diverse, and possibly more computationally efficient, than one using only 1-opt moves.

S1.2. Sequential and Stochastic Hero

As originally described, hero is a parameterless, steep ascent method which sequentially evaluates each decision variable and assigns it to the best state available at the time of evaluation [26]. Iteration continues until no improvement can be found (Listing S1). Since the decision variables are evaluated sequentially, an implicit dependency on the ordering of tree data exists. Maintaining this fixed ordering likely increases correlations between decisions made in different iterations. While such autocorrelation is presumably beneficial if the data can be advantageously ordered, it appears difficult to determine such orderings. A mitigation against potentially poor data ordering is therefore to randomize the order of evaluation in each iteration. We refer to this variation as stochastic hero (Listing S2) to distinguish it from the default implementation of hero using sequential evaluation.

For a stand represented by N trees, in both variations of hero one iteration calls the growth and yield models N times to evaluate N 1-opt flips. If the models are O(N), as Organon is, then hero's per iteration cost is $O(N^2)$. While hero does not guarantee a convergence rate, we have observed it to require approximately log(N) iterations, resulting in behavior similar to an $O(N^2 \log N)$ algorithm. Hero is therefore the least computationally expensive of the heuristics considered here. Hero is also attractive for its simplicity. It has no parameters to adjust and minimizes computation by automatically stopping once it has converged.

Listing S1. Pseudocode for hero's sequential implementation.

```
iterations ← 25 // never reached: 12-14 observed in practice

acceptedLEV ← randomize tree selection
foreach (iteration in 1...iterations) {
```

```
    foreach (tree in plot data) { // consider all trees on the plot      candidateLEV ← 1-opt flip of
tree
      if (candidateLEV > acceptedLEV) {
        acceptedLEV ← candidateLEV // accept move
      }
    }
    if (no moves accepted in iteration) {
      break; // stop if converged
    }
  }
```

Listing S2. Pseudocode's for hero's stochastic implementation. Changes from the sequential implementation are in bold.

```
iterations ← 25 // never reached: 9-12 observed in practice

acceptedLEV ← randomize tree selection
foreach (iteration in 1...iterations) {
  randomize order of all trees in plot data
  foreach (tree in randomized order) {
    candidateLEV ← 1-opt flip of tree
    if (candidateLEV > acceptedLEV) {
      acceptedLEV ← candidateLEV // accept move
    }
  }
  if (no moves accepted in iteration) {
    break; // stop if converged
  }
}
```

S1.3. Monte Carlo Heuristics: Simulated Annealing, Record-to-Record Travel, Threshold Accepting, and Great Deluge

Simulated annealing, record-to-record travel, threshold accepting, and great deluge are all Monte Carlo algorithms which randomly select trees to evaluate [27–29]. Where these four heuristics differ is in acceptance of moves which decrease LEV. Simulated annealing uses probabilistic acceptance of such disimproving moves, great deluge accepts any move whose LEV is above a rising "water" level, threshold accepting accepts any move whose LEV exceeds a variable fraction of the previously accepted LEV, and record-to-record travel accepts all moves with LEVs above a reactive threshold. High probabilities, low water levels, small fractions, and low thresholds result in most or all moves being accepted. In keeping with the simulated annealing analogy, these are referred to as hot states. Since such permissive move acceptance reduces or eliminates convergence towards an improved solution, all four heuristics use cooling schedules to apply increasing selection pressure to moves. Numerous schedules have been proposed [40] §4.1 and cooling frequently reaches cold states where only improving moves are accepted. The move acceptance rate can also be monitored and, once improving moves become too difficult to find, cooling can be restarted from a newly hot state in an attempt to locate a different solution which improves upon the solutions already observed. As with cooling schedules, many such reheating schedules have been demonstrated [41] §3.3.1.

Due to the cost of growth and yield calculations, hot states' low move efficiency is undesirable in individual tree selection. We therefore begin Monte Carlo runs in a fully cold state and reheat once a maximum LEV had been reached. Because moves diverge from a maxima orders of magnitude more rapidly than they converge we limit LEV decreases in reheats to restrict the computational effort required to reconverge.

To determine move acceptance in simulated annealing, we use a running average to adapt to the stand structure and LEV. While the averaging was inspired by Goh et al. [42], our use differs in most details (Listing S3). We implement extended great deluge with variable water level similar to McMullan [43] and, for threshold accepting, move acceptance remains as defined by Dueck and Scheuer [29] (Listings S5 and S6). While

threshold accepting is often used with a linear or geometric progression of thresholds and a fixed number of iterations per threshold, we use a small number of thresholds with a variable number of iterations per threshold for simplicity in configuring reheating. Threshold accepting has also been configured for adaptive cooling and reheating [44] but only fixed reheating was used here.

We were unsuccessful in locating previous use of reheating with record-to-record travel, though heating necessarily increases the deviation allowed from the best solution. While Dueck [28] did not define a cooling schedule for record-to-record travel, we found reheats had vanishingly low probabilities of reaching improving solutions without cooling. We therefore added a geometric cooling parameter similar to simulated annealing's temperature reduction parameter α. The resulting variant of record-to-record travel is somewhat simpler than the other three Monte Carlo heuristics and appears to provide equivalent functionality (Listing S4).

In addition to use of 1-opt flips we also evaluated reactively switching to 2-opt exchanges when a search appeared to be close to a LEV maximum. Since this use of exchanges produced negligible improvement, the results we present for Monte Carlo heuristics use exclusively 1-opt flips. However, this study explored only a limited subset of the many possible combinations of 1-opt moves, 2-opt moves, and reheats.

Listing S3. Simulated annealing with reheating. N is the number of trees used to represent the stand. Key distinctions from other Monte Carlo heuristics are in bold.

```
α ← 0.7 // cooling parameter, rate depends on iterationsPerTemperature
acceptanceProbability ← 0 // cold start
iterations ← 19N
iterationsPerTemperature ← 10
movesHighestIncreasedOrReheat ← 0
runningAverage ← -infinity
runningAverageRetention ← 10
reheatAfter ← integer(1.6N)
reheatBy ← 0.33; // probability, depends on α

acceptedLEV ← randomize tree selection
highestLEV ← acceptedLEV
foreach (iteration in 1...iterations) {
  foreach (iteration in 1...iterationsPerTemperature) {
    randomly select one tree from all trees on the plot
    candidateLEV ← 1-opt flip of selected tree
    LEVincrease ← candidateLEV – currentLEV
    relativeIncrease ← LEVincrease / runningAverage
    ++ movesSinceHighestIncreasedOrReheat

    if (candidateLEV > acceptedLEV) or
       (random probability in [0, 1] < exp(ln(acceptanceProbability) * relativeIncrease) {
      acceptedLEV ← candidate LEV // accept move
      if (acceptedLEV > highestLEV) {
        highestLEV ← acceptedLEV
        movesSinceHighestIncreasedOrReheat ← 0
      }

      if (runningAverage == -infinity) {
        runningAverage ← LEVincrease // initialize moving average
      }
      else {
        runningAverage ← LEVincrease / runningAverageRetention + (1 – 1 / runningAver-
ageRetention) * runningAverage
      }
    }

    if (movesSinceAcceptOrReheat > reheatAfter) {
```

```
        acceptanceProbability ← min(acceptanceProbability + reheatBy, 1.0)
        movesSinceHighestIncreasedOrReheat ← 0
      }
    }
    acceptanceProbability ← α * acceptanceProbability
  }
```

Listing S4. Record-to-record travel with reheating. Key distinctions from other Monte Carlo heuristics are in bold.

```
α ← 0.75 // reheat cooling parameter
deviation ← 0 // cold start
increaseAfter ← integer(1.6N)
relativeIncrease ← 0.0075 // controls exploration range
iterations ← 19N

acceptedLEV ← randomize tree selection
bestLEV ← acceptedLEV
foreach (iteration in 1...iterations) {
  randomly select one tree from all trees on the plot
  candidateLEV ← 1-opt flip of selected tree
  deviation ← α * deviation
  if (candidateLEV > acceptedLEV - deviation) {
    acceptedLEV ← candidateLEV // accept move

    if (acceptedLEV > bestLEV) {
      bestLEV ← acceptedLEV
    }
  }

  if (iteration == increaseAfter) { // one time reheat
    deviation ← relativeIncrease * |bestLEV| // absolute value
  }
}
```

Listing S5. Threshold accepting with an initial convergence followed by one or more reheats. Key distinctions from other Monte Carlo heuristics are in bold.

```
thresholds ← { 1.0, 0.999, 1.0 [, 0.999, 1.0, ...] } // of arbitrary length
iterationsPerThreshold ← integer({ 11.5N, 25, 7.5N [, 25, 7.5N, ...] }) // matches thresholds, varied from default: see table A4

acceptedLEV ← randomize tree selection
foreach (threshold in thresholds) {
  foreach (iteration in 1...iterationsPerThreshold[threshold]) {
    randomly select one tree from all trees on the plot
    candidateLEV ← 1-opt flip of selected tree
    if (candidateLEV > threshold * acceptedLEV) {
      acceptedLEV ← candidateLEV
    }
  }
}
```

Listing S6. Great deluge with reheating. Key distinctions from other Monte Carlo heuristics are in bold.

```
finalMultiplier ← 1.75 // can depend on LEV improvement over run
initialMultiplier ← 1.25 // can depend on LEV improvement over run
iterations ← 19N
movesSinceAcceptOrReheat ← 0
lowerWaterAfter ← integer(1.6N)
lowerWaterBy ← 0.0033; // controls exploration range
```

```
acceptedLEV ← randomize tree selection
bestLEV ← acceptedLEV
waterLevel ← initialMultiplier * initialLEV
rainRate ← (finalMultiplier * initialLEV – waterLevel) // iterations
foreach (iteration in 1...iterations) {
  randomly select one tree from all trees on the plot
  candidateLEV ← 1-opt flip of selected tree

  if (candidateLEV > acceptedLEV) or (candidateLEV > waterLevel) {
    acceptedLEV ← candidateLEV // accept move
    movesSinceAcceptOrReheat ← 0

    if (acceptedLEV > bestLEV) {
      bestLEV ← acceptedLEV
    }
  }
  else {
    ++movesSinceAcceptOrReheat
  }

  if (movesSinceAcceptOrReheat > lowerWaterAfter) {
    waterLevel ← (1.0 – lowerWaterBy) * acceptedLEV
    movesSinceAcceptOrReheat ← 0
  }
}
```

S1.4. Steady State Genetic Algorithm with CD/RW Replacement

A genetic algorithm is expected to require more computation than hero or Monte Carlo heuristics but also to be more robust in solution quality [37]. Numerous genetic algorithms have been proposed [45], all of which breed new generations of solutions from a population. Selection pressure is applied through replacement of individuals in the population by offspring with higher LEV and possibly through other mechanisms [46,47]. We found the steady state (μ+λ) genetic algorithm using contribution of diversity and worst replacement (CD/RW SSGA) of Lozano et al. [30] produced more rapid convergence to lower variance than generational (μ,λ) algorithms or simply replacing the worst individual in the population. We also found uniform crossovers converged more quickly and with lower variance than 1- or 2-point crossovers. We therefore present results for a uniform crossover CD/RW SSGA.

Genetic algorithms rely on recombination and mutation of genetic material to make offspring different from their parents. We treated the harvest selections for the N trees representing a stand as a single chromosome. If the position of trees within chromosomes is not randomized, an implicit dependency on the ordering of tree data occurs when using k-point crossovers that is analogous to the dependency found in hero sequential. However, the benefits of randomizing 1- and 2-point crossovers against data ordering appeared to be small compared to the convergence advantage of a uniform crossover.

Mutations were performed by applying 1-opt or 2-opt exchange moves to each of the two children resulting from a breeding with the two mutation types occurring at independent probabilities. Experimentation found convergence was most rapid when mutation probabilities were inversely proportional to the LEV distance from convergence. Since nonlinear regression found lower quartile, median, and upper quartile LEV distances were well approximated by an exponential decline (bias < 0.0012% and root mean square error < 0.70% of the LEV range from population initialization to convergence), we present results for mutation schedules with a $1 - e^{-kx}$ form (Listing S7).

Listing S7. Steady state genetic algorithm (SSGA) using uniform crossover and CD/RW (contribution of diversity and replace worst) replacement of individuals within its population [30] Fig. 3, along with variable 1- and 2-opt mutation probabilities. Generation and assessment of child is considered to be a move.

```
maximumGenerations ← integer(5.5N^0.6)
populationSize ← 30 // controls convergence and solution quality

// thin intensity = { 0.5/P, 1.5/P, ... , (P - 0.5)/P}
initialPopulation ← randomTreeSelection(intensity = { 0.5, 1.5, ..., populationSize – 0.5}/populationSize)
currentLEVs ← lev(initialPopulation) // list of each individual's LEV
nextLEVs ← currentLEVs // next generation
foreach (generation in 1...generations) {
  foreach (breeding in 1...populationSize) {
    parent1 ← currentLEVs[random individual] // uniform probability
    parent2 ← currentLEVs[random individual] // allows selfing

    // uniform crossover on an individual tree basis
    foreach (tree in plot data) {
      firstSelection ← parent1[tree]
      secondSelection ← parent2[tree]
      if (parent1[tree] != parent2[tree]) {
        buffer ← firstSelection
        if (random probability < 0.5) {
          firstSelection ← secondSelection
        }
        if (random probability < 0.5) {
          secondSelection ← buffer
        }
      }
      child1[tree] ← firstSelection
      child2[tree] ← secondSelection
    }

    1optProbability ← 1 - exp(-8 * generation/generations)
    2optProbability ← 0.1 * 1optProbability
    if (random probability < 1optProbability) { // child 1
      1optMutation(child1, random tree)
    }
    if (random probability < 2optProbability) {
      2optMutation(child1, 2 random trees)
    }
    child1LEV ← lev(child1)
    if (min(nextLEVs) < child1LEV) {
      nextLEVs[CD/RW replacement] ← child1LEV
    }

    if (random probability < 1optProbability) { // child 2
      1optMutation(child2, random tree)
    }
    if (random probability < 2optProbability) {
      2optMutation(child2, 2 random trees)
    }
    child2LEV ← lev(child2)
    if (min(nextLEVs) < child2LEV) {
      nextLEVs[CD/RW replacement] ← child2LEV
    }
  }

  currentLEVs ← nextLEVs
}
```

S1.5. Tabu Search

The eighth and final heuristic we evaluated was tabu search [31]. Like hero, tabu sequentially evaluates trees and picks the best move. Unlike hero, each iteration within a tabu search checks the selections of all N trees and then accepts only the single move which yielding highest available LEV. Tabu is therefore an $O(N^3 / \log N)$ algorithm and the most computationally expensive heuristic considered here: each growth and yield calculation is O(N), complete evaluation of each move's 1-opt neighborhood performs N growth and yield calculations, and about N / log N moves are required to converge on scheduling N trees. High move costs are a well-known limitation of tabu search [31]. One mitigation is to split the problem into smaller subsets which can be solved more efficiently. However, we were unable to identify any partitioning based on random subsets, height classes, or similar or dissimilar diameter quantiles which meaningfully reduced computation while maintaining solution quality. Results are therefore presented for a full neighborhood form of tabu search randomized only by the initial tree selection (Listing S8).

Also unlike hero, which is parameterless, tabu search accepts a single tenure parameter. While tenure encourages diversification by prohibiting moves from being undone, a fixed tenure of length L permits return to a given solution after L + 1 or more moves. A tabu search can therefore stagnate due to oscillation about a local maximum. Variable tenure lengths and reactive escape mechanisms are sometimes used to suppress such oscillations [41] §4.2. Within our computational capacity, exploration suggested use of variable tenure and escapes slightly increased median LEVs during extended tabu searches but both mechanisms appeared detrimental at practical search durations due to 1–5% increases in variance. We therefore present results for fixed tenures without escape. We also observed pseudooscillations where trees were replaced by similar trees. Since pseudooscillation approximates a 2-opt exchange and may be more efficient than searching 2-opt neighborhoods directly we did not attempt to suppress pseudooscillation.

Listing S8. Tabu search.

```
iterations ← integer(4.25N / log(N))
maxTenure ← integer(0.1N) // move tenures are set to a random value in [2, maxTenure]
tabuList ← { 0, 0, ... }

acceptedLEV ← randomize tree selection
bestLEV ← acceptedLEV
foreach (iteration in 1...iterations) {
  bestNonTabuIterationLEV ← -infinity
  bestNonTabuSchedule ← null
  bestNonTabuTree ← null
  bestTabuIterationLEV ← -infinity
  bestTabuSchedule ← null
  bestTabuTree ← null
  foreach (tree in plot data) { // full neighborhood evaluation
    [candidateLEV, candidateSchedule] ← 1-opt flip of tree
    isTabu ← tabuList[tree][candidateSchedule] > 0
    if (candidateLEV > bestLEV) {
      bestNonTabuIterationLEV ← candidateLEV // aspirational criteria
      bestNonTabuSchedule ← candidateSchedule
      bestNonTabuTree ← tree
    }
    else if (isTabu == false) and (candidateLEV > bestNonTabuIterationLEV) {
      bestNonTabuIterationLEV ← candidateLEV
      bestNonTabuSchedule ← candidateSchedule
      bestNonTabuTree ← tree
    }
    else if (candidateLEV > bestTabuIterationLEV) {
      bestTabuIterationLEV ← candidateLEV
      bestTabuSchedule ← candidateSchedule
      bestTabuTree ← tree
```

```
      }

      foreach (schedule in { thin, final harvest }) {
        tabuList[tree][schedule] ← max(tabuList[tree][schedule] - 1, 0)
      }
    }

    if (bestNonTabuIterationLEV != -infinity) {
      acceptedLEV ← bestNonTabuIterationLEV // accept non-tabu move
      tabuList[bestNonTabuTree][bestNonTabuSchedule] ← random(2, maxTenure)
    }
    else if (bestTabuIterationLEV != -infinity) {
      acceptedLEV ← bestTabuIterationLEV // forced to accept tabu move
      tabuList[bestTabuTree][bestTabuSchedule] ← random(2, maxTenure)
    }
  }
```

S1.6. Conventional Prescription Optimization Using Complete Enumeration

We provide this listing for clarity in details of tree selection. While complete enumeration is conceptually simple, implementations can differ in how the combined intensity of thinning from above, proportionally, and below is bounded and in how those intensities become quantized when applied to individual trees. Implementations may also differ in how specific trees are selected for proportional thinning.

Listing S9. Simplified implementation of complete enumeration as used for conventional prescription optimization in this study. More complex stepping logic is helpful when the minimum and maximum intensities are not integer multiples of the step size and avoids the continue and break statements.

```
minimumIntensity ← 30 // % basal area, inclusive
maximumIntensity ← 90 // % basal area, inclusive
step ← 1 // %

bestLEV ← -infinity
[bestBelow, bestProportional, bestAbove] ← -1
foreach (below in 0...100 by step) { // 0%, 1%, 2%, ...
  foreach (proportional in 0...100 by step) { // 0%, 1%, 2%, ...
    foreach (above in 0...100 by step) { // 0%, 1%, 2%, ...
      if (below + proportional + above < minimumIntensity) {
        continue
      }
      else if (below + proportional + above > maximumIntensity) {
        break
      }

      sortedTrees ← sort trees from smallest to largest DBH
      candidateSchedule ← no trees selected
      // select trees from above or below until basal area limit exceeded
      // Rounds up (away from zero) to nearest whole tree removal.
      candidateSchedule ← selectLargestDiameterTrees(above, sortedTrees)
      candidateSchedule ← selectSmallestDiameterTrees(below, sortedTrees)
      // uses a floating point accumulator to handle fractional steps
      // For example, 40% proportional selects 2 out of 5 trees using a
      // repeating 1, 0, 0, 1, 0 pattern.
      candidateSchedule ← selectEveryNthTree(proportional, sortedTrees)

      candidateLEV ← growthAndYield(candidateSchedule)
      if (candidateLEV > bestLEV) {
        bestLEV ← candidateLEV
        bestBelow ← below
        bestProportional ← proportional
```

```
                bestAbove ← above
            }
        }
    }
}
```

S1.7. Parameterization and Initialization

While heuristics respond to their implementation and parameterization, we did not find exact choices to be critical. In most cases, parameters differing 10–25% from those given in Listings S1–8 exhibited limited change in the LEV distribution or amount of computation required. We also could have, for example, used reheating to allow hero stochastic to move beyond the first local optima encountered. When evaluating parameter effects or the LEV results reported here, each heuristic run began with unique, random values of $\mathbf{u}$ (Eqs. 2 and 3). For the seven hill climbing heuristics—tabu search, both variants of hero, and the four Monte Carlo heuristics—each tree in $\mathbf{u_0}$ had a 50% probability of being selected for thinning. This random restart method forms $\mathbf{u_0}$ as an uninformative prior for thinning, resulting in populations of LEVs which are independent and identically distributed. The genetic algorithm's population of solutions was initialized with 30 choices of $\mathbf{u}$, $\mathbf{u_0}$…$\mathbf{u_{29}}$, each with a linear increase in a tree's probability of removal (Listing S7). We found distributing $\mathbf{u_0}$…$\mathbf{u_{29}}$ across different levels of thinning intensity in multiple diameter classes, diameter quantiles, or height quantiles nearly always decreased the genetic algorithm's convergence rate.

S2. Supplemental Results: Heuristic Convergence

Figures 1–5 in the main text aggregate results from 552 combinations of plot, thinning age, rotation length, and heuristic. Since each combination is associated with a convergence probability distribution, we present distributions for 24 of the combinations in figures S1–3 below. In our experience, monitoring these distributions is helpful when designing or parameterizing heuristics and they provide valuable context when comparing results of different publications or heuristic behavior on different problems.

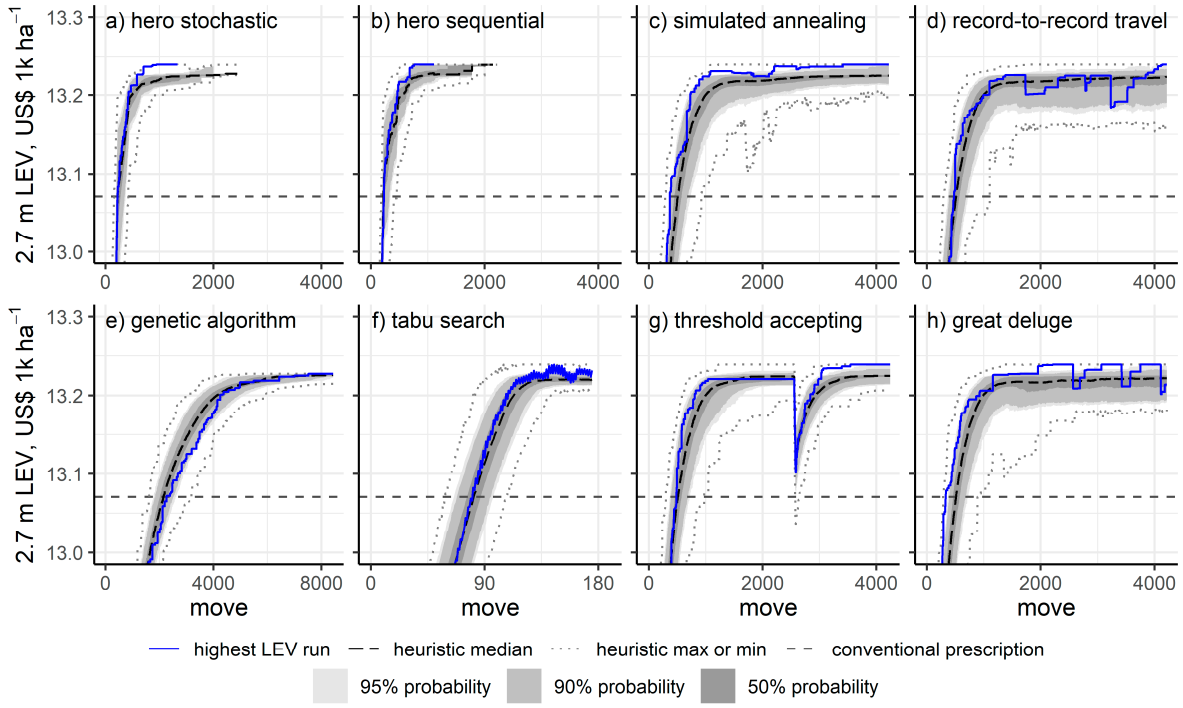

Figure S1. LEV distributions as a function of move and heuristic on the 2.7 m plot for thinning at 40 years with a 70 year rotation and 1000 runs per heuristic. Horizontal dashed lines indicate the LEV of the optimized conventional prescription. Note that reheating widens the Monte Carlo heuristics' distributions and that the genetic algorithm's x-axis is extended and tabu search's x-axis shortened.

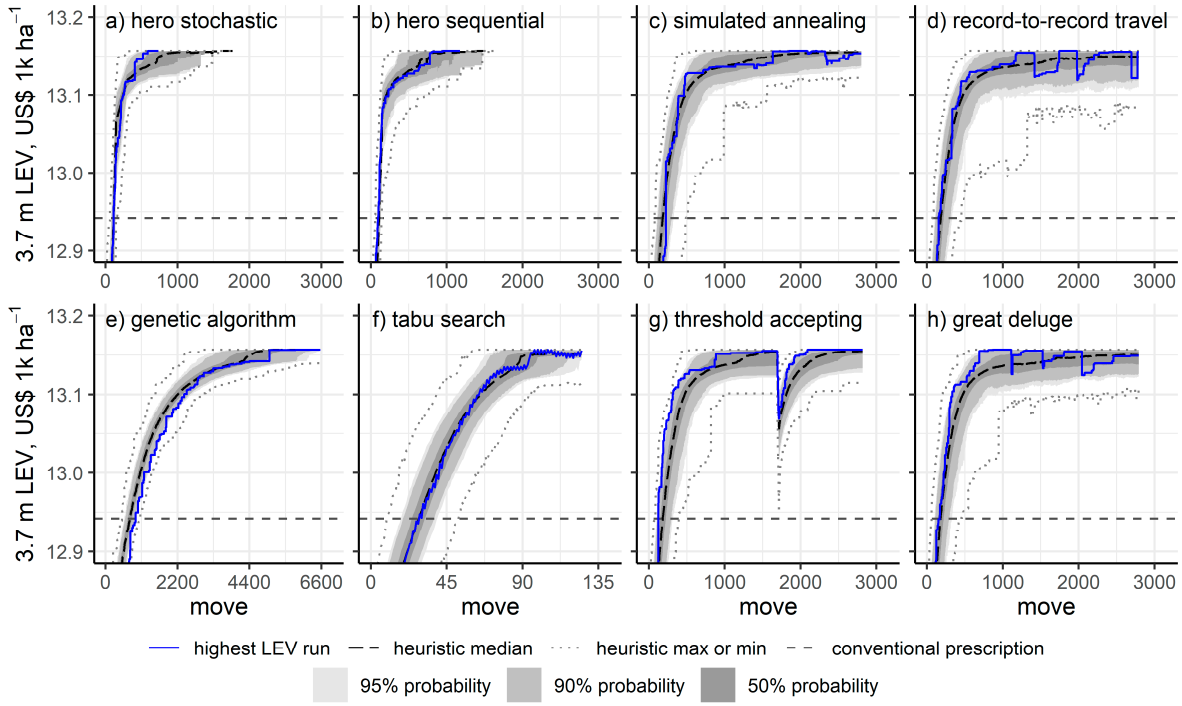

Figure S2. LEV distributions as a function of move and heuristic on the 3.7 m plot for thinning at 40 years with a 65 year rotation and 1000 runs per heuristic. As in Figure S1, the x-axis is extended for the genetic algorithm and shortened for tabu search.

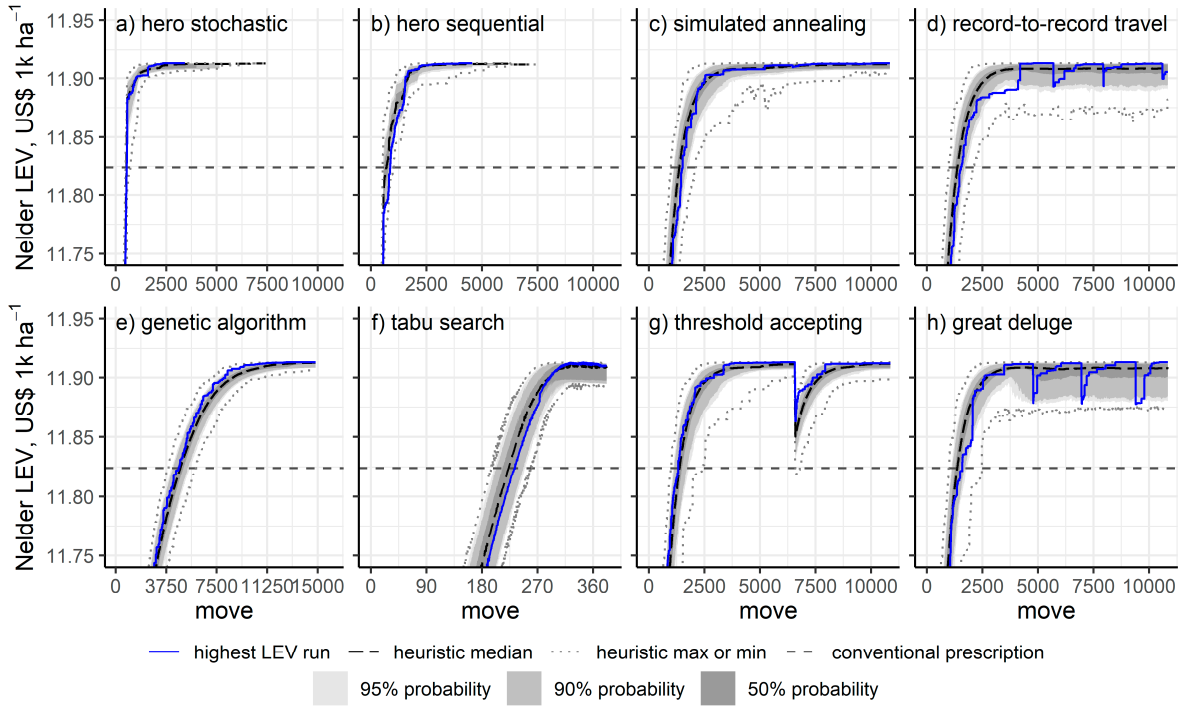

Figure S3. LEV distributions as a function of move and heuristic for the Nelder plot for thinning at 40 years with a 65 year rotation. 1000 runs were used for all heuristics except tabu search, where 100 runs were used. Also, as in Figures S1 and S2, the genetic algorithm and tabu search's x-axes are adjusted.

As shown in Figure 3 of the main text, all eight heuristics produce narrower ΔLEV distributions on the Nelder plot than on the 2.7 and 3.7 m plots. This result suggests the range of ΔLEV may tend to narrow as N increases, consistent with the additional result Figure S4 provides from merging the 2.7 and 3.7 m plots. Trees on the Nelder plot are also smaller (Table 3) than those on 2.7 and 3.7 m plots and are more diverse in size (diameter

standard deviation σ_{DBH} = 8.0 cm versus σ_{DBH} = 5.5–5.7 cm, σ_{height} = 3.9 m versus 2.3–3.0 m). Both of these factors may also act to narrow the Nelder plot's ΔLEV distribution.

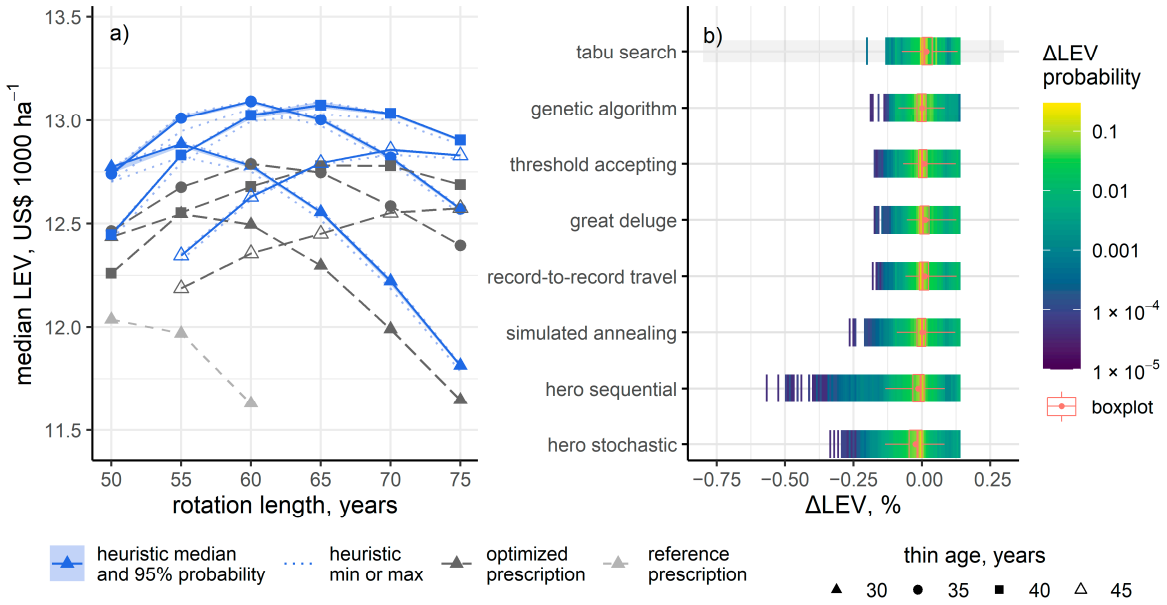

Figure S4. a) LEVs and b) ΔLEVs obtained from optimizing the 2.7 and 3.7 m plots as a merged plot with 369 trees rather than optimizing each plot separately. The merged plot has an average spacing of 3.2 m, similar to the 3.2–3.4 m spacings commonly used for plantation Douglas-fir in our study area (planting at 890–990 TPH). As with the Nelder plot, 1000 runs were performed for each heuristic except for using 100 tabu runs. 2.9 million prescriptions were evaluated in conventional prescription optimization.

S3. Software and Regression Imputation Used

Heuristics were implemented in C# and the CIPS-R 2.2.4 version of Organon was ported from Fortran to C#. C# code was built using Visual Studio 2019 Community Edition 16.8.3 x64 targets for .NET 5.0 and executed under Powershell Core 7.1.1 using Visual Studio Code 1.53.1. Source code is available under the GNU Public License from https://github.com/OSU-MARS. Data logged from C# was analyzed in R 4.0.3 using the cowplot (1.1.0), data.table (1.13.2), dplyr (1.0.2), forcats (0.5.0), fst (0.9.4), ggplot2 (3.3.2), magrittr (1.5), reshape2 (1.4.4), scales (1.1.1), stringr (1.4.0), and tidyr (1.1.2) packages. Regression imputation also used readr (1.4.0) and readxl (1.3.1).

Because heights were not measured for four trees on the Nelder plot, we used regression imputation to estimate their heights based on the 567 trees which were measured. The best supported, age-specific regression we identified for the plot, on both an adjusted R^2 and Akaike Information Criterion basis, was

$$H = 1.436 + 7.4315 \log(DBH) - 0.11692(spacing)^2 \quad \text{(S1)}$$

where H is the tree height in meters, DBH the tree's measured diameter at breast height in centimeters, and the type Ia [48] arc spacing in meters is given in Table 9 of [19]. We assumed arcs 1 and 2 had the same spacing as each other, arcs 16 and 17 also had an identical spacing, and that type Ia spacings occurred in reverse order for trees in the type Ib portion of the plot. This regression has an adjusted R^2 of 0.58 and both predictors are significant at $p < 0.001$.